

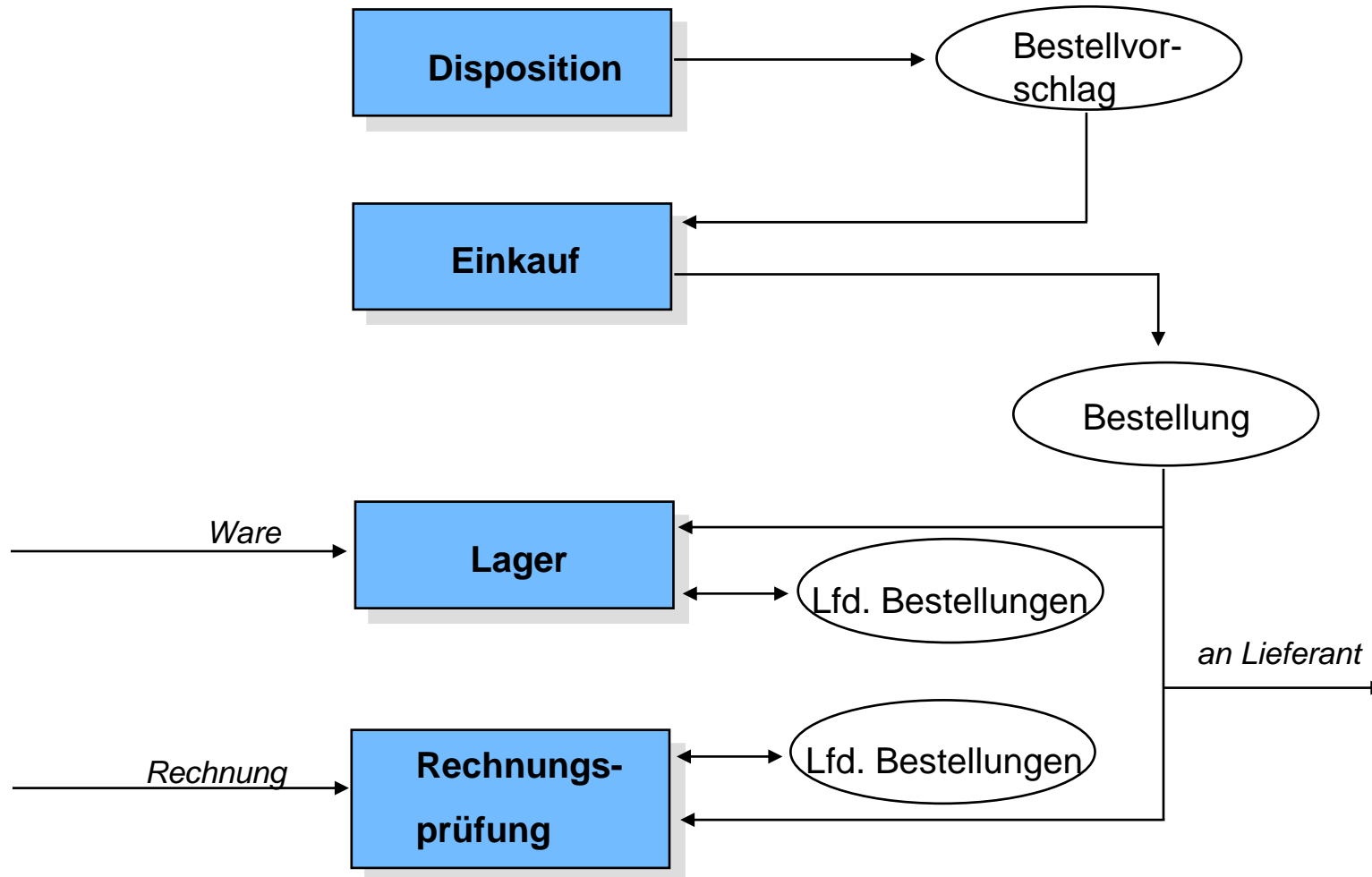
DATABASES

Autor: Walter Rothlin (18.4.2023)

Datenhaltung

Vorname	Nachname	Strasse	PLZ	Ort	Abteilung	Standort	Vorgesetzter
Max	Meier	Maibach	8855	Wangen	E&F	Uster	Ernst Blass
Hans	Müller	Grafenried 1	8854	Siebnen	Verkauf	Uster	Ernst Blass
Fritz	Jenny	Blumenweg 88	8855	Wangen	HR	Uster	Ernst Blass
Paul	Vogt	Blumenweg 88	8855	Nuolen	HR	Zürich	Fritz Jenny
Johannes	Mächler	Mythenblick	8853	Lachen	Einkauf	Zürich	Ernst Blass
Kurt	Grafenried	Bachstr. 8	7000	Bern	Einkauf	Zürich	Johannes Mächler
Heinz	Vogt	Schafmatt 33	3412	Biel	E&F	Uster	Max Meier
Claudia	Schättin	Etzelstr. 7	8808	Horgen	E&F	Uster	Max Meier

Gemeinsame Daten



Auswirkungen der IT auf die Organisation

Häufige Probleme traditioneller Abläufe:

- Physischer Dokumententransport
- Kopien von Daten/Dokumenten werden für die Kommunikation benötigt
- **Redundantes, mehrfaches Halten von Daten**
 - Anfällig bei Datenänderungen
 - Eine Gesamtsicht auf die Daten fehlt

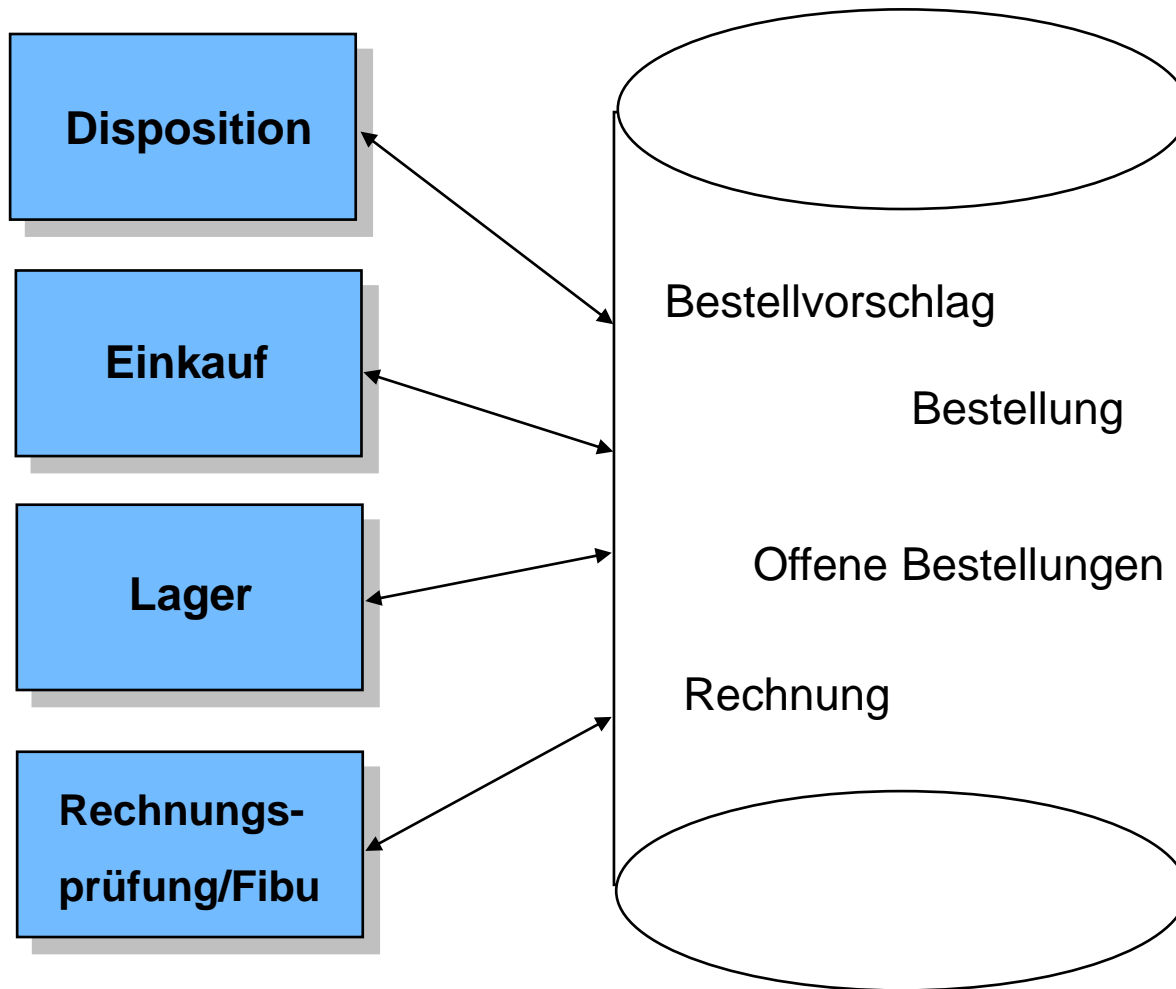
Lösung:

- Datenintegration: alle beteiligten Aufgaben greifen auf den **identischen** Datenbestand zu.
- Technisch realisiert durch zentrale Datenbanksysteme (wie z.B. IBM DB/2 , MS-Access, Oracle, MySql,...)

Datenhaltung – Persistenz → OO-Sicht

- Applikation arbeitet mit Objects
- Die Applikation ruft Objekt-Methoden und verändert ev. Zustände (Werte der Instanz-Variablen)
- Die Applikation greift auf Properties/Instanz-Variablen zu (über Getters, da Instanz-Variable private)
- Die Objekte befinden sich im Arbeitsspeicher und sind nach Programm-Ende verloren → Persistenz
- serializing / deserializing → Byte-Stream (nur Java, nicht lesbar)
- CSV, JSON, XML,...

Datenintegration



Daten Anwender Ebene (Interface):

- Views
- Stored-Procedures
- User-Berechtigungen

Administrator Ebene (Interface):

- Views
- Stored-Procedures
- User-Berechtigungen

DML (Data Manipulation Language)

DDL (Data Definition Language)

konzeptionelles Ebene (Datenmodell):

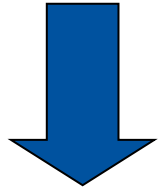
- Identifizieren des relevanten Informationsbedarfs (Attribute)
- Dabei: Identifizieren von Entitätstypen und Beziehungstypen.
- Zuordnen der Attribute zu Entitätstypen.
- Festlegen möglicher Attributwerte, Vorschläge für identifizierende Attribute.
- Bestimmen der Beziehungskardinalität.

Physisches Ebene (Persistence):

- Interne Strukturen in Files und Dateien
- Hersteller-Abhängig

Datenentwurf

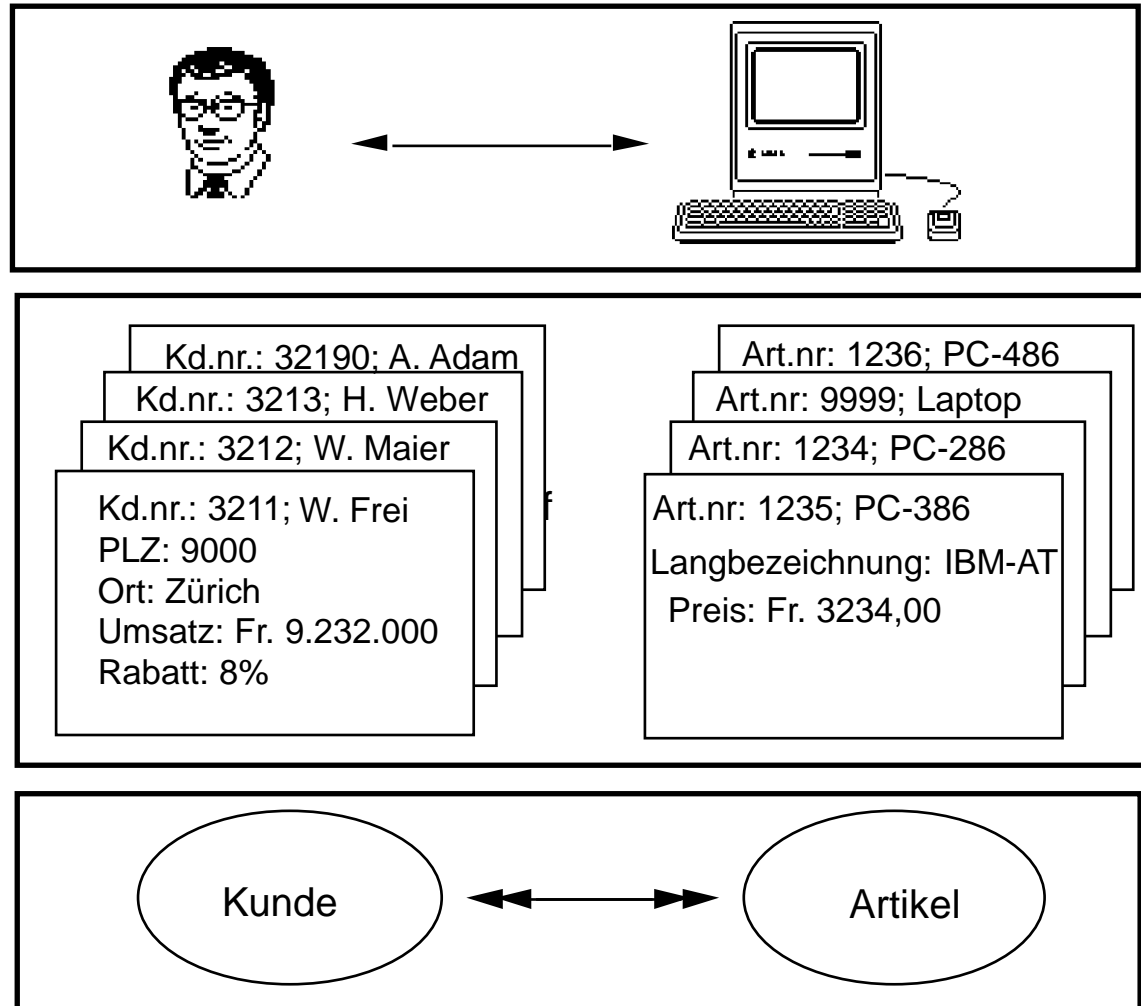
Reale Welt



Identifizierung
von Informations-
mengen



Beschreibung
der Realität mit
ER-Diagramm



Attribute von Entitätsmengen

Entitätsmengen haben Attribute

Ein Attribut ist eine informationelle Eigenschaft einer Entität, welches diese näher beschreibt.

Beispiel:

Attribute des Entitätstyps „Kunde“ sind u.a.

- Kundennummer (künstlich geschaffen; kein Bezug zur realen Welt)
- Name
- Vorname
- Adresse
- Kreditlimit

Modellierung betrieblicher Daten

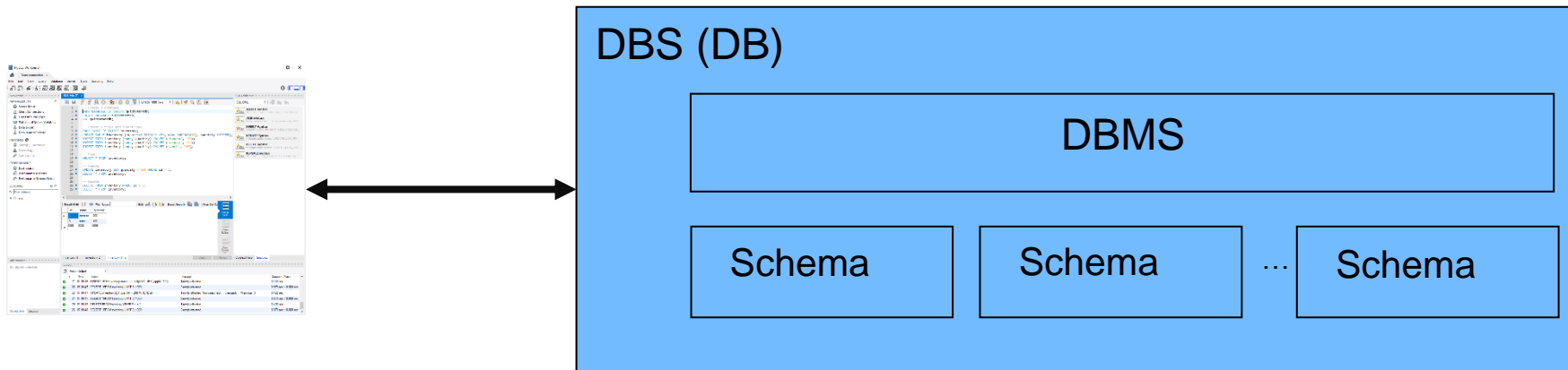
Aufgabe der Datenmodellierung ist es, für eine betriebliche Aufgabenstellung die fachlich benötigten

- Informationsobjekte (z.B. Kunden, Lieferanten)
- ihre Beziehungen und Abhängigkeiten
- und die wesentlichen Attribute/Eigenschaften der Informationsobjekte (z.B. Name, Adresse, Kreditlimit)

zu identifizieren und festzulegen.

Ziel der Datenmodellierung ist die Beschreibung von Anforderungen an eine zu realisierende Organisationslösung aus der Sicht der Daten.

Begriffe



Ein Datenbanksystem besteht aus ein oder mehreren Datenbanken und einem Datenbankmanagementsystem.

Eine Datenbank (Schema) ist eine Sammlung von strukturierten, inhaltlich zusammengehörigen Daten.

Ein Datenbankmanagementsystem (DBMS) besteht aus Programmen zum Aufbau, zur Kontrolle und Änderung von Daten.

Workbench ist Client für DBMS (meistens über Netz zugreifbar)

Begriffsbestimmungen

Informationsobjekt ("Entität", "Entity") → **Tables**

Eine Entität ist ein individuelles Objekt der realen Welt (z.B. Kunde Maier in Zürich) (z.B. Adresse, Umsatz, Kreditlimit). Da in einem Informationssystem nicht nur ein, sondern mehrere Kunden gespeichert werden sollen, wird die Summe aller einzelnen Entitäten auch "Entitätsmenge" genannt.

Eigenschaften, Merkmale von Entitäten → **Attributes** (Spalten der Tables)
z.B. Name, Vorname, Artikelbezeichnung,...

NULL → undefinierter Attribute-Wert (außerhalb des Wertebereichs)

- 0, Nullstring, Null-Pointer sind nicht NULL
- Ob ein Attribute **NOT NULL** sein kann, wird als Meta-Attribute für jedes Attribute definiert.

Tables, Attributes, Records

Attribute, Spalte, Column

Dataset, Record, Row,
Entity-Instance, Tupel

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250

Table / Tabelle

<https://www.youtube.com/watch?v=-fQ-bRIlhXc> (ERD)

Begriffsbestimmungen

Identifikation einer Entität → **Primär Schlüssel, Primary Key**

Ein Attribute oder eine Menge von Attributes (zusammengesetzter Schlüssel), welches die Entität eindeutig identifiziert. z.B. AHV-Nr, Auftrags-Nr,...

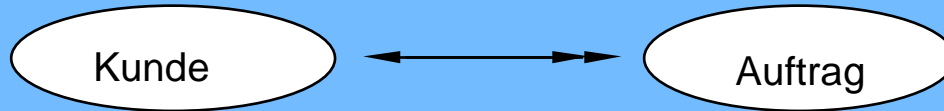
- Eindeutig (innerhalb der Tabelle)
- Laufend zuteilbar (keine begrenzte Menge)
- Möglichst kurz und nicht zusammengesetzt
- Invariant (Verändert sich über die Lebensdauer nicht) → alte AHV bei Heirat

Beziehung (**Relation**) → zwischen zwei Tables z.B. Ein Student hat eine Telefonnummer. Im ERD eine Verbindungslinie zwischen Tables.

Cardinality: Min / Max

ERD (Cardinality)

Beispiel der Kardinalität



- . Zu einem Kunden existieren mehrere (0-n) Aufträge
- . Zu jedem Auftrag gibt es genau einen Kunden

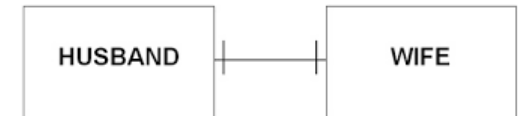
- Jede Abteilung besitzt zwischen 5 und 20 Mitarbeitern
- Jeder Mitarbeiter gehört zu höchstens einer Abteilung



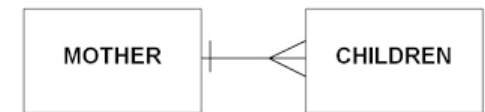
UML

Symbol	Cardinality	Meaning	Alternative Symbole
	1	One - Mandatory	
	1..n 1..*	Many - Mandatory	
	0..1	One - Optional	
	0..n 0..*	Many - Optional	

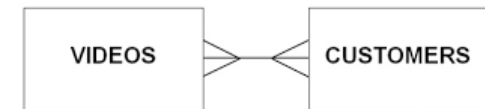
One-to-one relationship



One-to-many (or many-to-one) relationships

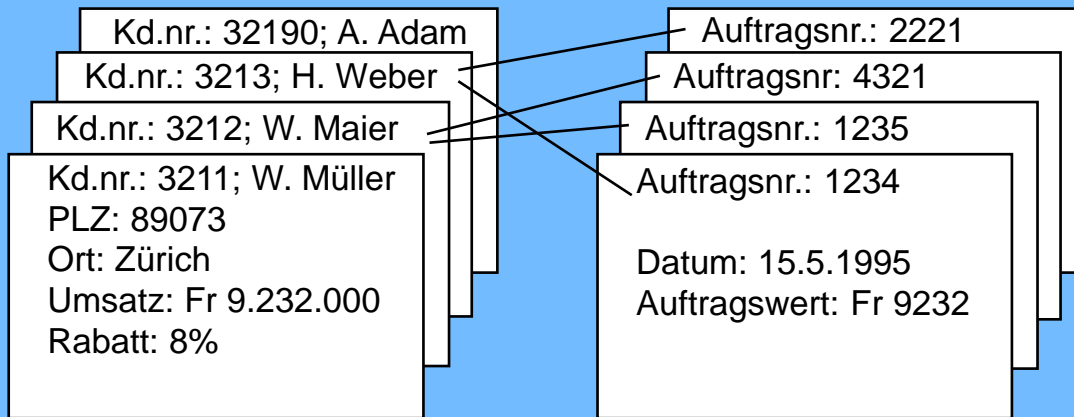


Many-to-many relationships

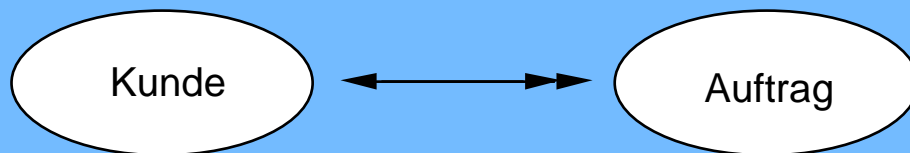


Beziehung 1:n

Realität



Beschreibung der Realität



- . Zu einem Kunden existieren mehrere (0-n) Aufträge
- . Zu jedem Auftrag gibt es genau einen Kunden

Angestellter

0..n

1

Abteilung

0..n: Eine Abteilung kann auch keine Angestellte haben

1: Ein Angestellter gehört immer zu einer Abteilung (keine freien MA)

Angestellter			Abteilung	
PersNr	Name	AbtNr	AbtNr	Name
1001	Marxer	1	1	Verkauf
1002	Widmer	2	2	Marketing
1003	Steiner	3	3	Entwicklung
1010	Affolter	1	4	Finanzen
1050	Widmer	3	5	QS
1100	Meier	1		

Referentielle Integrität

Angestellter		
PersNr	Name	AbtNr
1001	Marxer	1
1002	Widmer	2
1003	Steiner	3
1010	Affolter	1
1050	Widmer	3
1100	Meier	1

Abteilung	
AbtNr	Name
1	Verkauf
2	Marketing
3	Entwicklung
4	Finanzen
5	QS

Referenzielle Integrität: Wertebereich von **Angestellter.AbtNr** ist die Menge der vorhandenen Werte von **Abteilung.AbtNr** (dynamisch)

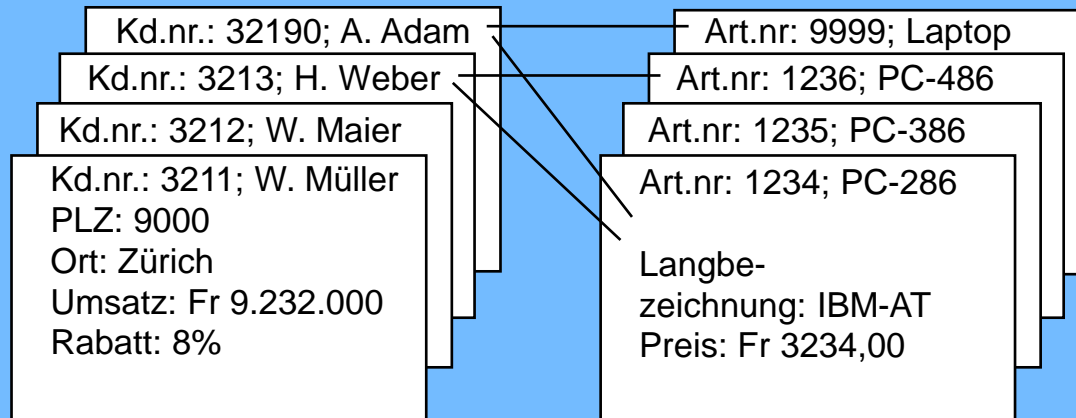
Kritische Operationen:

Angestellter: Insert, Update AbtNr (AbtNr muss bereits in **Abteilung** vorhanden sein)

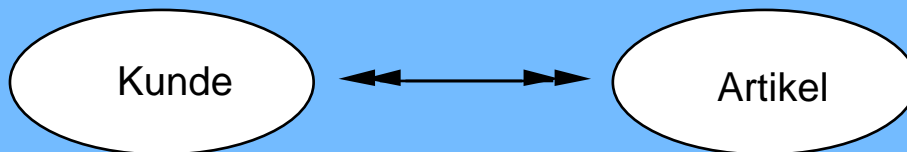
Abteilung: Delete, Update AbtNr (AbtNr darf nicht mehr in **Angestellter** vorhanden sein)

Beziehung m:n

Realität

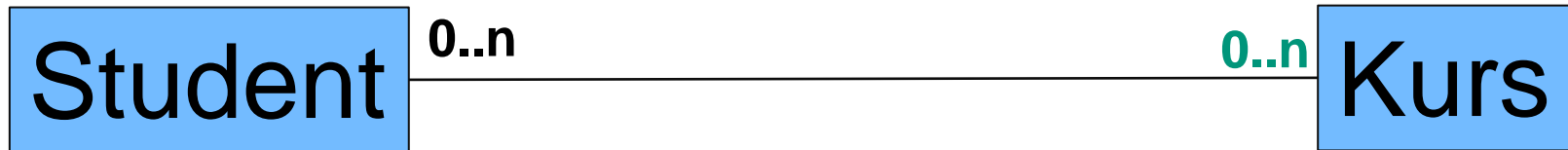


Beschreibung der Realität



- . Ein Kunde kauft mehrere (0-n) Artikel
- . Zu einem Artikel gibt es mehrere (0-n) Kunden

Beziehung m:n



0..n: Ein Kurs hat mehrere angemeldete Studenten

0..n: Ein Student belegt mehrere Kurse

Student		Belegung		Kurs	
PersNr	Name	PersNr	KursNr	KursNr	Bezeichnung
1001	Marxer	1001	1	1	Java
1002	Widmer	1001	2	2	Datenbank
1003	Steiner	1002	1	3	C#
1004	Affolter	1002	2	4	JEE
1050	Widmer	1004	1	5	Perl
1100	Meier				

Diagram illustrating the m:n relationship between Student and Kurs using the Belegung table as a junction. Arrows show the following connections:

- Student 1001 (Marxer) is connected to Belegung rows (1001, 1) and (1001, 2).
- Student 1002 (Widmer) is connected to Belegung rows (1001, 2) and (1002, 1).
- Student 1003 (Steiner) is connected to Belegung row (1002, 1).
- Student 1004 (Affolter) is connected to Belegung row (1002, 2).
- Student 1050 (Widmer) is connected to Belegung row (1004, 1).
- Belegung row (1001, 1) is connected to Kurs row 1 (Java).
- Belegung row (1001, 2) is connected to Kurs row 2 (Datenbank).
- Belegung row (1002, 1) is connected to Kurs row 3 (C#).
- Belegung row (1002, 2) is connected to Kurs row 4 (JEE).
- Belegung row (1004, 1) is connected to Kurs row 5 (Perl).

Primärschlüssel

Primärschlüssel sind spezielle Attribute mit folgenden impliziten Eigenschaften (Constraints):

- Unique
- Not NULL

Arten von Primärschlüsseln:

- **Einfacher (Eindeutiger) Primärschlüssel**
 - Ein Attribut allein ist Primärschlüssel
 - i.d.R. ein „künstlicher“ Schlüssel, der eindeutig ein Objekt der Tabelle identifizieren kann (Surrogate Key)
- **Zusammengesetzter Primärschlüssel**
 - Mehrere Attribute zusammen sind Primärschlüssel
 - So viele Attribute wie erforderlich, um zusammen eindeutig ein Objekt der Tabelle zu identifizieren



Structured Query Language SQL

[ɛskjuːˈɛl], [ˈsiːkwəl]

DML: Data Manipulation Language

Befehle zur Datenmanipulation

CRUD → Create Read Uppdate Deleate

```
SELECT Postleitzahl, Stadt FROM Kunden WHERE Stadt='Zürich' ORDER BY Postleitzahl;
```

```
INSERT INTO Student (MatrNr, Name) VALUES (27124, 'Schulz'), (27125, 'Schmidt');
```

```
UPDATE Personal SET Gehalt=Gehalt*2, Strasse='Poststr.1' WHERE Abteilung='EDV';
```

```
DELETE FROM Bestellungen WHERE Bestellstatus IS NULL;
```

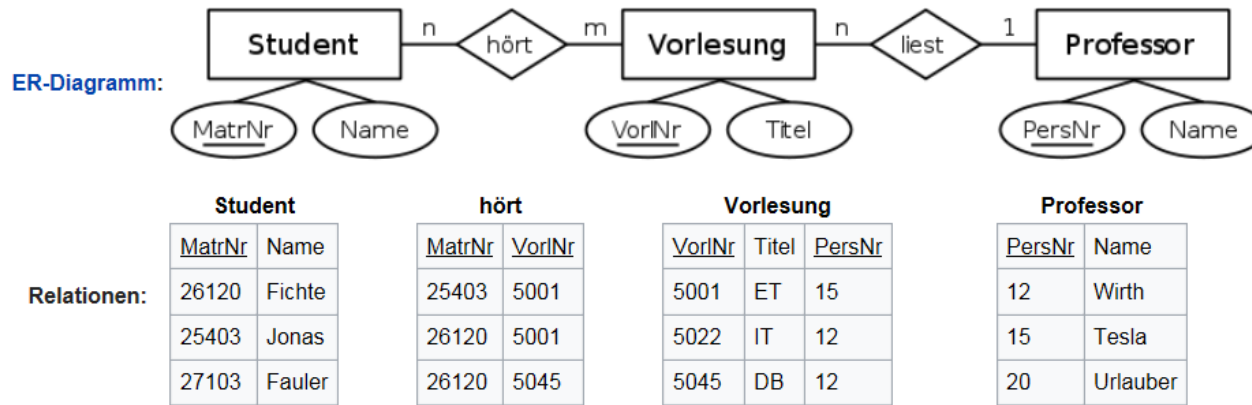
SQL: select...

<https://dev.mysql.com/doc/refman/5.7/en/retrieving-data.html>
<https://dev.mysql.com/doc/refman/5.7/en/pattern-matching.html>

```
SELECT DISTINCT
    FIELD_NAME      AS fName,
    FIELD_NAME_1    AS fName1
FROM TAB_NAME
WHERE
    FIELD_NAME      = 'MONTAG'                or  -- not case-sensitive
    FIELD_NAME      regexp binary '^Dienstag$' or  -- case-sensitive
    FIELD_NAME      like 'Mo%'                 or  -- % wildcard (0 .. n)
    FIELD_NAME      like '__i'                 OR  -- _ placeholder (genau 1)
    FIELD_NAME      is not NULL                 or
    FIELD_NAME      not in ('A','D')            or  -- Menge
    FIELD_NAME1     not between 8 and 10        and  -- Intervall
ORDER BY
    FIELD_NAME DESC,
    FIELD_NAME1
GROUP BY FIELD_NAME;
```

-- aufsteigend is Default
-- zusammenfassen

SQL: select...



-- einfache Abfrage

```
SELECT
    Name                                AS Familienname,
    MatrNr                              AS Personal_Nummer,
    DATE_FORMAT(last_update, '%Y-%M-%d') AS LastUpdate
FROM
    Student
WHERE
    ((MatrNr >= 26120) AND (Name LIKE '%a%')) -- % → 0..n      _ → 1 Zeichen
ORDER BY
    Name    ASC,
    MatrNr  DESC;
```

SQL: UNION, INTERSECTION, MINUS

```
select FIELD from TAB1      -- Vereinigungsmenge
union all
select FIELD from TAB2;
```

```
select FIELD from TAB1      -- Schnittmenge
intersect
select FIELD from TAB2;
```

```
select FIELD from TAB1      -- Differences
minus
select FIELD from TAB2;
```

DML: Data Manipulation Language

Insert

```
INSERT INTO Student (MatrNr, Name) VALUES (27124, 'Schulz'), (27125, 'Schmidt');
```

```
INSERT INTO TAB_NAME (sysdate) VALUES (TO_DATE('05-AUG-1960', 'DD-MON-YYYY'));
```

```
INSERT INTO Student (MatrNr, Name) SELECT MatrNr, Name FROM StudentOld;
```

DML: Data Manipulation Language

Update

```
UPDATE Personal SET  
    Gehalt = Gehalt*2,  
    Strasse = 'Peterliwiese 33'  
WHERE Abteilung='EDV';
```

DML: Data Manipulation Language

Delete

```
DELETE FROM Bestellungen  
WHERE bestellstatus IS NULL;  -- ohne Where-Clause, alle Records werden gelöscht!
```

```
TRUNCATE TABLE Student;  -- reset von auto increment Werten. Ist ein DDL Command!
```


DML: Data Manipulation Language

Insert, Update, Delete (CRUD)

```
INSERT INTO language (name) VALUES ('Schweizerdeutsch'), ('Dutch');
```

```
SELECT * FROM language; -- Was für eine ID hat Schweizerdeutsch bekommen?
```

```
UPDATE film SET original_language_id=1 WHERE film_id=1;
```

```
UPDATE film SET original_language_id=1 WHERE film_id=2;
```

```
SELECT
    f.title           AS Title,
    lang.name         AS Sprache,
    orgLang.name      AS Originalsprache
FROM
    film AS f
INNER JOIN language AS lang ON f.language_id = lang.language_id
LEFT OUTER JOIN language AS orgLang ON f.original_language_id = orgLang.language_id;
```

```
DELETE FROM language WHERE name='Schweizerdeutsch';
```

```
UPDATE film SET original_language_id=null WHERE film_id IN (1,2);
```

Normalisieren / Joins

Benutzersicht (Views) vs. Frei von Redundanzen

Vorname	Nachname	Strasse	PLZ	Ort	Abteilung	Standort	Vorgesetzter
Max	Meier	Maibach	8855	Wangen	E&F	Uster	Ernst Blass
Hans	Müller	Grafenried 1	8854	Siebnen	Verkauf	Uster	Ernst Blass
Fritz	Jenny	Blumenweg 88	8855	Wangen	HR	Uster	Ernst Blass
Paul	Vogt	Blumenweg 88	8855	Nuolen	HR	Zürich	Fritz Jenny
Johannes	Mächler	Mythenblick	8853	Lachen	Einkauf	Zürich	Ernst Blass
Kurt	Grafenried	Bachstr. 8	7000	Bern	Einkauf	Zürich	Johannes Mächler
Heinz	Vogt	Schafmatt 33	3412	Biel	E&F	Uster	Max Meier
Claudia	Schättin	Etzelstr. 7	8808	Horgen	E&F	Uster	Max Meier

- Dies ist z.B. die Sicht auf die Mitarbeiter vom Lohnbüro aus ==> Viele Redundanzen
- Wieso sind Redundanzen «schlecht»?
 - Bei Änderungen / Mutationen wird es schwierig, nichts zu vergessen!
- Wie finde ich Redundanzen?
 - Doppelte Einträge auslagern und referenzieren reicht nicht
 - Business Aspekt muss berücksichtigt werden
- Wie eliminiere ich Redundanzen?
 - Normalisierung, in mehrere Tabellen aufteilen und referenzieren
- Wie können nach Normalisierung die Benutzersichten wieder hergestellt werden?
 - Joins und Views

Normalisierung

- 1.NF (erste Normalform): <https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/erste-normalform/>
Eine Tabelle ist in der 1. NF, wenn ihre Felder nur einfache Werte (keine mengenmässigen Werte) annehmen können

CD_Lied

CD_ID	Album	Jahr der Gründung	Titelliste
4711	Anastacia - Not That Kind	1999	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}
4712	Pink Floyd – Wish You Were Here	1964	{1. Shine On You Crazy Diamond}
4713	Anastacia - Freak of Nature	1999	{1. Paid my Dues}



CD_Lied

CD_ID	Albumtitel	Interpret	Jahr der Gründung	Track	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

Normalisierung

- 2.NF (zweite Normalform): <https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/zweite-normalform/>
Eine Tabelle ist in der 2. NF, wenn sie in 1.NF ist und jede Spalte vom Primärschlüssel voll funktional abhängig ist.

CD_Lied

CD_ID	Albumtitel	Interpret	Jahr der Gründung	Track	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

CD_Lied (inkonsistent)

CD_ID	Albumtitel	Interpret	Jahr der Gründung	Track	Titel
4711	I Don't Mind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues



CD

CD_ID	Albumtitel	Interpret	Jahr der Gründung
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1964
4713	Freak of Nature	Anastacia	1999

Lied

CD_ID	Track	Titel
4711	1	Not That Kind
4711	2	I'm Outta Love
4711	3	Cowboys & Kisses
4712	1	Shine On You Crazy Diamond
4713	1	Paid my Dues

Normalisierung

- 3.NF (dritte Normalform): <https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/dritte-normalform/>
Eine Tabelle ist in der 3. NF, wenn sie in der 1. + 2. NF ist und keine transitiven Abhängigkeiten aufweist (d.h. sie ist nicht weiter zerlegbar nach dem Schema der 2. NF)

CD

CD_ID	Albumtitel	Interpret	Jahr der Gründung
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1964
4713	Freak of Nature	Anastacia	1999



CD

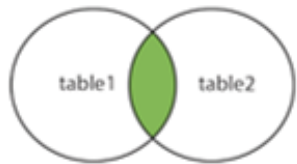
CD_ID	Albumtitel	Interpret_ID
4711	Not That Kind	311
4712	Wish You Were Here	312
4713	Freak of Nature	311

Künstler

Interpret_ID	Interpret	Jahr der Gründung
311	Anastacia	1999
312	Pink Floyd	1964

Joins

INNER JOIN



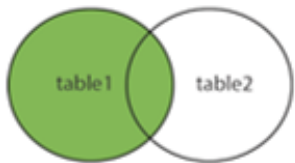
```
-- inner join auf language
select film.title, language.name
from film
inner join language on film.language_id = language.language_id; -- 1000
```

```
-- inner join auf original_language
select film.title, language.name
from film
inner join language on film.original_language_id = language.language_id; -- 0
```

Links: Haupttabelle
(von dort will ich alle Rows)
Rechts: Zusatztabelle
(Leserichtung!)

```
-- welche Filme haben eine original_language gesetzt?
select film.title
from film
where film.original_language_id <> null;
```

LEFT JOIN



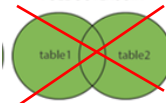
```
-- Ich will eine Liste aller filmemit der original_language auch wenn ein Film kein original language gesetzt hat
-- Left Join ist natürlicher (normaler) outer Join
select film.title, language.name
from film
left outer join language on film.original_language_id = language.language_id; -- 1000
```

RIGHT JOIN



```
-- Ich will eine Liste aller filmemit der original_language auch wenn ein Film keinen original language gesetzt hat
-- Right Join ist invertierter Join
select film.title, language.name
from language
right outer join film on film.original_language_id = language.language_id; -- 1000
```

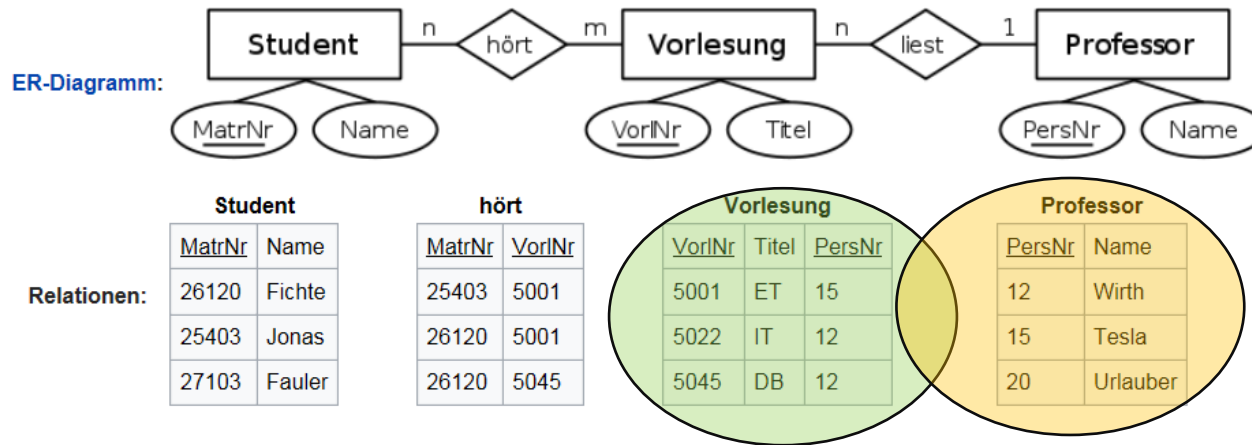
FULL OUTER JOIN



In MySQL nicht implementiert!

Cross Joins → Kartesisches Produkt

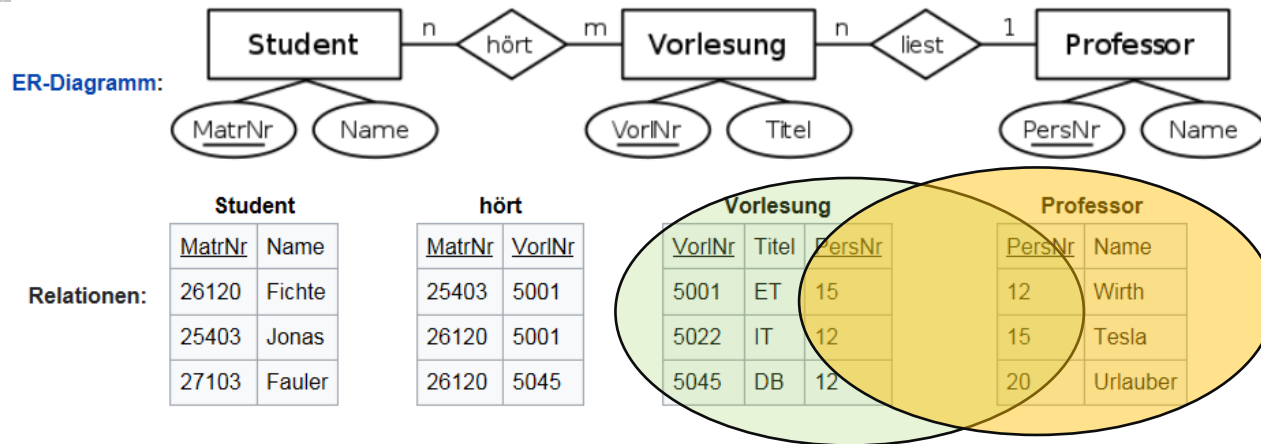
SQL: select...



-- Join (inner): Professoren ohne Vorlesung und Vorlesungen ohne Professor werden damit nicht angezeigt.

```
SELECT Vorlesung.VorlNr, Vorlesung.Titel, Professor.PersNr, Professor.Name
FROM Professor
INNER JOIN Vorlesung ON Professor.PersNr = Vorlesung.PersNr;
```

SQL: Outer Join

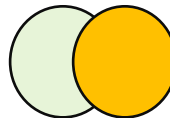


// Join (outer): Professoren ohne Vorlesung sind enthalten

```
SELECT Professor.PersNr, Professor.Name
FROM Professor
LEFT OUTER JOIN Vorlesung ON Professor.PersNr = Vorlesung.PersNr;
```

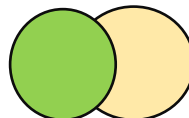
-- Alle Professoren

```
FROM Professor LEFT OUTER JOIN Vorlesung
FROM Vorlesung RIGHT OUTER JOIN Professor
```



-- Alle Vorlesungen

```
FROM Vorlesung LEFT OUTER JOIN Professor
FROM Professor RIGHT OUTER JOIN Vorlesung
```



-- Alle Vorlesungen und alle Professoren

```
FROM Professor FULL OUTER JOIN Vorlesung
```

-- Old Way for Outer joins! (don't use it)

-- =====

```
select A.Name, A.Age, B.Skill
from WORKER A, WORKERSKILL B
where A.Name = B.Name(+)
order by A.Name
```

SQL: View

Virtuelle Tabelle, basiert auf Tabellen oder anderen Views, wird mit CREATE VIEW erzeugt und mit einer Select-Anweisung (abgefüllt) definiert.

```
DROP VIEW IF EXISTS film_sprachen;
CREATE VIEW film_sprachen AS
    SELECT
        f.title           AS Title,
        lang.name         AS Sprache,
        orgLang.name      AS Originalsprache
    FROM
        film AS f
    INNER JOIN language AS lang      ON f.language_id      = lang.language_id
    LEFT  JOIN language AS orgLang  ON f.original_language_id = orgLang.language_id;

select * from film_sprachen;
```

Gründe für eine View:

- Information Hiding
- Security
- Decoupling

Transaktion

Problem: Kontoübertrag:

Sparkonto



Saldo: 2'000.-

Lohnkonto



Saldo: 5'000.-

Vermögen: 7'000.-



Übertrag: 1'234.50

Transaktion

Problem: Kontoübertrag:

Transaktion besteht es mehreren (in diesem Fall aus zwei)
Operationen: Abbuchen, Einbuchen

Der Konto-Besitzer muss immer sein korrektes Vermögen sehen!

Wem gehört Geld während dem Transport?

Was passiert mit Geld, falls dies nicht eingebucht werden kann?

Transaktion

Konsistenz: (semantische Datenintegrität) Es darf keine Widersprüche geben

Transaktion: Konsistenz-erhaltende Operation

ACID:

ATOMICITY (Alles oder nichts)

CONSISTENCY (Konsistenzerhaltung)

ISOLATION (Isolierter Ablauf, Independence)

DURABILITY (Dauerhaftigkeit, Persistenz)

Start –Transaction → Commit / Rollback

BEGIN TRANSACTION

DELETE FROM...

INSERT INTO...

ROLLBACK TRANSACTION

Backup / Recovery

REDO: Sämtliche Transaktionen, welche seit dem letzten CHECKPOINT COMMITTED wurden

UNDO: Sämtliche Transaktionen, welche noch nicht COMMITTED wurden

Backup der DB:

- Full backup

- Incremental backup

- Mirroring

Rekonstruktion nach DB Verlust:

- Rückspielen des letzten Backups

- REDO der Transaktionen seit dem letzten Backup

Functions

-- Bei Fehlermeldung SET ausführen!

```
SET GLOBAL log_bin_trust_function_creators = 1;
```

```
DROP FUNCTION IF EXISTS HelloFct;
```

```
Delimiter //
```

```
CREATE FUNCTION HelloFct(p_input_string CHAR(20)) RETURNS CHAR(50)
```

```
  BEGIN
```

```
    RETURN concat('Hallo: ', p_input_string);
```

```
  END
```

```
//
```

-- Verwendung:

```
select HelloFct('Walti') as HALLO;
```


Stored-Procedures

```
DROP PROCEDURE IF EXISTS test_searchCountry;
Delimiter //
CREATE PROCEDURE test_searchCountry(
    IN searchQuery VARCHAR(20),
    IN caseSensitive BOOLEAN)
BEGIN
    IF caseSensitive
    THEN SELECT country AS Land
        FROM country
        WHERE country LIKE BINARY searchQuery;
    ELSE SELECT country AS Land
        FROM country
        WHERE country LIKE searchQuery;
    END IF;
END//
```

-- Verwendung:

```
call test_searchCountry('Germany', false);
call test_searchCountry('Germany', true);
call test_searchCountry('Germany', false);
call test_searchCountry('Germany', true);
```

<http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx>

DDL: Data Definition Language

Befehle zur Definition der Strukturen (Meta-Data)

Details über Tabellen und Schematas

-- <https://dev.mysql.com/doc/refman/5.7/en/tables-table.html>

```
SELECT
    table_schema,
    table_name,
    table_type
FROM    INFORMATION_SCHEMA.TABLES
WHERE   table_name LIKE '%film%'
ORDER BY table_schema, table_name, table_type;
```

-- <https://dev.mysql.com/doc/refman/5.7/en/columns-table.html>

```
SELECT
    table_schema,
    table_name,
    column_name,
    DATA_TYPE,
    IS_NULLABLE,
    COLUMN_DEFAULT
FROM    INFORMATION_SCHEMA.COLUMNS
WHERE   table_schema = 'sakila'
ORDER BY table_schema , table_name, column_name;
```

DDL: Data Definition Language

Befehle zur Definition der Strukturen (Meta-Data)

CREATE SCHEMA

```
CREATE SCHEMA `schema_name`;  
USE `schema_name`;
```

Create Table

```
DROP TABLE IF EXISTS tab_name;  
CREATE TABLE tab_name (  
    id            INT            NOT NULL AUTO_INCREMENT,  
    field_1       VARCHAR(45)    NOT NULL,  
    field_2       VARCHAR(5)     NOT NULL,  
    field_3       INT(4)         UNSIGNED NULL  
                        DEFAULT 5,  
    last_update  TIMESTAMP      NOT NULL  
                        DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    PRIMARY KEY (id),  
    UNIQUE INDEX field_1_UNIQUE (field_1 ASC)  
);
```

Table definition ändern

```
ALTER TABLE tab_name ADD(FIELD3    DATE);  
ALTER TABLE tab_name MODIFY (FIELD3 DATE NOT NULL);
```

Table löschen

```
DROP TABLE tab_name;
```

SCHEMA löschen

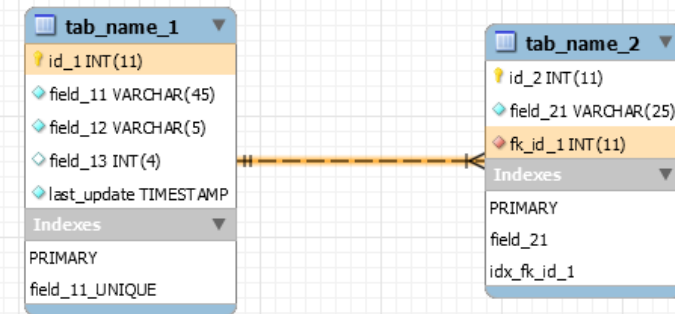
```
DROP SCHEMA `schema_name`;  
DROP DATABASE `schema_name`;
```

SQL Konsistenzbedingungen

```
DROP TABLE IF EXISTS tab_name_1;
CREATE TABLE tab_name_1 (
  id_1          INT          NOT NULL AUTO_INCREMENT,
  field_11      VARCHAR(45)  NOT NULL,
  field_12      VARCHAR(5)   NOT NULL,
  field_13      INT(4)        UNSIGNED NULL DEFAULT 5,
  last_update   TIMESTAMP    NOT NULL
                                DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id_1),
  UNIQUE INDEX field_11_UNIQUE (field_11 ASC)
);
```

```
DROP TABLE IF EXISTS tab_name_2;
CREATE TABLE tab_name_2 (
  id_2          INT          NOT NULL AUTO_INCREMENT,
  field_21      VARCHAR(25)  NOT NULL UNIQUE,          -- Sekundärschlüssel
  fk_id_1       INT          NOT NULL,
  KEY idx_fk_id_1 (fk_id_1),
  CONSTRAINT fk_tab_name_2__tab_name_1
    FOREIGN KEY (fk_id_1) REFERENCES tab_name_1 (id_1)
    ON DELETE RESTRICT ON UPDATE CASCADE,

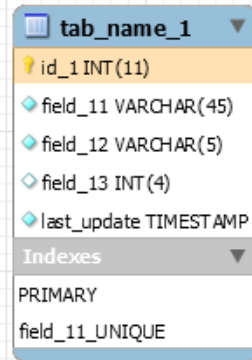
  PRIMARY KEY (id_2));
```



SQL Referenzielle Integrität

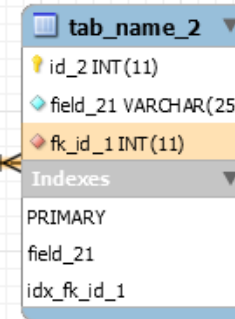
TAB_NAME_1

Referenzierte Tabelle



TAB_NAME_2

Haupttabelle



Wird in **TAB_NAME_1** ein **Tupel_1** gelöscht (oder aKey geändert), müssen alle Tuples in **TAB_NAME_2**, welche das **Tupel_1** referenzieren (via **fk_id_1**) “geändert” werden:

- ON DELETE CASCADE
- ON DELETE RESTRICT (Default)
- ON DELETE SET NULL
- ON DELETE SET DEFAULT
- ON UPDATE CASCADE
- ON UPDATE RESTRICT (Default)
- ON UPDATE SET NULL
- ON UPDATE SET DEFAULT

Tuples in **TAB_NAME_2** werden auch gelöscht

Tupel_1 wird nicht gelöscht (meistens bevorzugt)

fk_id_1 wird NULL gesetzt

fk_id_1 wird auf den Default-Wert gesetzt

fk_id_1 wird aktualisiert (meistens bevorzugt)

Tupel_1 wird nicht upgated

fk_id_1 wird NULL gesetzt

fk_id_1 wird auf den Default-Wert gesetzt

DB-Connection in Python

```
import mysql.connector

print("Connecting to sakila....", end="", flush=True)
mydb = mysql.connector.connect(
    host      = "localhost",
    user      = "root",
    passwd    = "admin",
    database  = "sakila"
)
print("completed!")
```

DB-Abfrage in Python

```
stm_selectCities = "SELECT * FROM city"

stm_selectCities = """
SELECT
    city_id      AS ID,
    city         AS Name,
    country_id   AS Country
FROM
    city
WHERE
    city like '0%'
"""

mycursor = mydb.cursor()
mycursor.execute(stm_selectCities)
myresult = mycursor.fetchall()
```

Result-Set in Python

```
print("+-----+-----+-----+-----+")
print("| Id      | City                               | Country ID |")
print("+-----+-----+-----+-----+")
for aRec in myresult:
    print("| {plh:4d} |".format(plh=aRec[0]), end="")
    print(" {plh:30s} |".format(plh=aRec[1]), end="")
    print(" {plh:10d} |".format(plh=aRec[2]), end="")
    print()
    print("+-----+-----+-----+-----+")
print("Records found:", len(myresult), myresult)
```